ARMY RESEARCH LABORATORY

A Two-Dimensional Finite Difference Algorithm on the KSR1

by Christopher S. Kenyon

ARL-MR-213

April 1995

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	April 1995	Final, from July 1993 to September 1994

4. TITLE AND SUBTITLE
A Two-Dimensional Finite Difference Algorithm on the KSR1

5. FUNDING NUMBERS
PE: 62120

6. AUTHOR(S)
Christopher S. Kenyon

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
U.S. Army Research Laboratory
Attn: AMSRL-WT-ND
2800 Powder Mill Road
Adelphi, MD 20783-1197

8. PERFORMING ORGANIZATION REPORT NUMBER
ARL-MR-213

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
U.S. Army Research Laboratory
2800 Powder Mill Road
Adelphi, MD 20783-1197

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
AMS code: 612120.H250011
ARL PR: 4FE7E7

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Executing electromagnetic pulse (EMP) codes on the fastest possible computers has been a goal of research in recent years, because it would allow new classes of EMP problems to be solved. A new kind of supercomputer architecture has emerged in recent years, called a massively parallel processor (mpp). This study examines the performance of a two-dimensional finite difference algorithm, a Yee algorithm, which is a major part of some EMP codes, on an mpp computer, the KSR1. Performance of the Yee algorithm, using a mesh of 2000 × 2000, scales roughly in proportion to the number of processors, up to about 32. The maximum performance of the Yee algorithm on the KSR1 was 161 Mflops using 50 processors, or 5.8 times faster than on a 50-MHz IBM RS/6000 workstation. For a 2000 × 2000 matrix multiply test, representative of an important algorithm in moment methods for solving electromagnetic problems, the peak performance on the KSR1 was 279 Mflops with 32 processors. This matrix multiply ran about three times faster than on the IBM workstation. It was concluded that the modest gains found here were insufficient to warrant a full port of a large EMP code to the KSR1.

14. SUBJECT TERMS	15. NUMBER OF PAGES
KSR, Yee, MPP, parallel computer, EMP, finite difference	22
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

Contents

Figures

1. Introduction

A primary thrust in the development of new supercomputers has been in the creation of new massively parallel architectures. In 1993, the Department of Defense (DoD) acquired a KSR1 massively parallel computer for a site at the Army Research Laboratory (ARL).

The KSR1 has the potential to reach the computing speeds of today's fastest supercomputers. The ARL KSR1 was originally configured with 256 processors, or nodes, each capable of up to 40 Mflops, or 40 million floating point operations, per second. As a whole, the computer could potentially yield a maximum computing speed of 10 Gflops, or 10 billion floating point operations, per second. Each CPU node has a local cache of 32 MB of memory that it can share with all the other nodes resulting in a maximum of 8 GB of shared memory available. Thus, ARL's KSR1 has the potential to supply 8 GB of real memory to a single parallel task. Recently, ARL increased the number of nodes to 288 and has typically divided it into three machines, two with 128 nodes and one with 32. One of the 128-node machines, Pisa, uses a newer, less stable version of the KSR operating system and is designated as a development machine. Thus the tests of this evaluation were run on either the 128-node Eiffel machine or the 32-node London machine, both with the more stable 1.1.4.1 KSR operating system.

The object of this study was to measure how fast the KSR1 could run a specific two-dimensional (2-D) finite difference algorithm, useful in solving electromagnetic problems. Specifically, this algorithm, a 2-D version of the Yee algorithm [1], provides time-domain solutions of Maxwell's equations. The Yee algorithm models electromagnetic wave generation and propagation, as well as its scattering from and coupling to [2] objects and surfaces of interest. In particular, we have found the Yee algorithm useful in our electromagnetic pulse (EMP) environment prediction codes [3,4] because of its ability to adapt to certain time-varying parameters of the problems. Acceleration of the Yee algorithm by significantly more than a factor of 10 would be a critical step toward solving a more complex class of EMP environment prediction problems than the current state-of-the-art computer codes.

2. Creation of Parallel Computer Code

Migration of computer code to a parallel computer typically requires that parts of the code be rewritten to maximally, or even significantly, exploit the multiple processors. First, one needs to write the algorithms, which for this study were in FORTRAN (the KSR can automatically parallelize FORTRAN code; it cannot for C code) in a parallel form. The KSR provides three options for parallelizing the code. One has the option of parallelizing the code automatically using the preprocessor KAP, parallelizing it manually by inserting directives or calls to the run-time library, presto, or performing a combination of the manual and automatic approaches. In this third case, one inserts some specific directives and then allows KAP to parallelize other sections of the code as allowed.

Although not essential for maximizing parallel execution, a technique originally suggested by a KSR analyst also greatly improved the execution speed of the Yee code on the KSR. In this technique, the field arrays (B_ϕ, E_x, and E_z) were consolidated into one array, an X array, in which the first component was B_ϕ, and so forth for the other components.

The parallelization of the Yee code required that computation for the magnetic field B spatial loops had to be separated from the electric field E loops. This separation was necessary because the B field calculations were dependent upon the value of the E field at the time step before that just computed. This Yee algorithm rewrite was necessary in order for it to run on parallel computers. However, for the KSR, special command statements or compiler directives had to be inserted to keep the B loops separate through the FORTRAN compilation process. The appendix lists the FORTRAN source code used to run a simple 2-D version of the Yee algorithm in parallel on the KSR1 in this study.

For the performance runs, the separate field arrays of the Yee algorithm were consolidated into one field array. Two KSR directives, which appear as "C*KSR*...," were placed in the FORTRAN source code within the time loop. Then KAP created an intermediate file by adding the remaining directives needed to fully parallelize the program. Finally, the FORTRAN compiler created an executable file from this intermediate file using "O2" optimization and the options –para, –lpresto, and –lpmon. The first two options were necessary for linking the "pthreads," a sequential flow within a process, and also the parallel runtime library, presto. The third option was to link the timing libraries to the code. Additionally, –lksrblas was added to provide the BLAS (Basic Linear Algebra Subroutine) library for the matrix multiply tests.

3. Performance of a Parallel Yee Algorithm

Initially, in this study, after the FORTRAN for the EMP code BERM2 was rewritten for better parallel execution, KAP was run, or executed, on it. However, this parallel code never executed at better than about 1 Mflop, no matter how many processors were executing it. Using more processors actually slowed it down. This code was quite complicated compared to the simple Yee algorithm discussed in this report and included the Yee algorithm as its principal subroutine, as well as other subroutines with time-dependent current drivers and air conductivity. The complexity of the algorithms and the type of floating-point operations used were probably the cause of this low performance. Still, this was a disappointing performance, since it was over an order of magnitude slower than achievable on our IBM RS/6000 workstations. Although the KSR1 would execute our FORTRAN–77 codes transparently, it would not yield a high speed transparently.

Subsequently, the parallelization effort was narrowed to the Yee algorithm. The performance of this 2-D Yee algorithm would set a maximum on the possible performance gain that we could expect for our Yee-based EMP code.

The parallel version of the simple Yee code, created as discussed in section 2, had apparently unavoidable data dependencies, which may have degraded execution efficiency compared with loops without those dependencies. Also, separation of the E and B spatial loops introduced additional machine overhead to handle the additional parallel section. Consequently, this extra overhead increased the computation time by as much as a factor of 3 or 4.

Figure 1 shows how the computational speed of this 2-D Yee algorithm scales with the number of nodes used for two different sizes of spatial arrays, 1000×1000 and 2000×2000. Main Yee algorithm loops, rather than initialization or data output, were timed. Wall clock time was considered more significant than user times, because only the former was indicative of what an investigator can accomplish in a given time. Default *real* precision on the KSR1 is 8 bytes [5], which is comparable to double precision on other machines. This 8-byte precision was the precision used for all the computations in this study.

Figure 2 shows computational speed per node or the relative efficiency of a processor. Processor efficiency reaches a maximum of about 6 Mflops per node. When the larger array problem uses only a few nodes, its efficiency is low because the program requires more memory than the real memory available to it; thus, paging occurs, resulting in a considerable slowdown in the execution. The maximum computational speed for the main loops including time advancement is about 161 Mflops for the 2000×2000 array, as seen in figure 1.

Figure 1. Computational speed of basic 2-D Yee algorithm. *IBM* refers to computer runs on IBM RS/6000 model 560. *Wall* refers to wall clock time measured runs on KSR1. *User* refers to measurements inferred from time that KSR was running USER process algorithm. Peak speed of 77.0 Mflops occurs with 32 nodes for array of 1000×1000 cells, based on elapsed wall clock time. Peak speed of 160.9 Mflops occurs with 50 nodes for array of 2000×2000 cells. FORTRAN was compiled with O2 optimization.

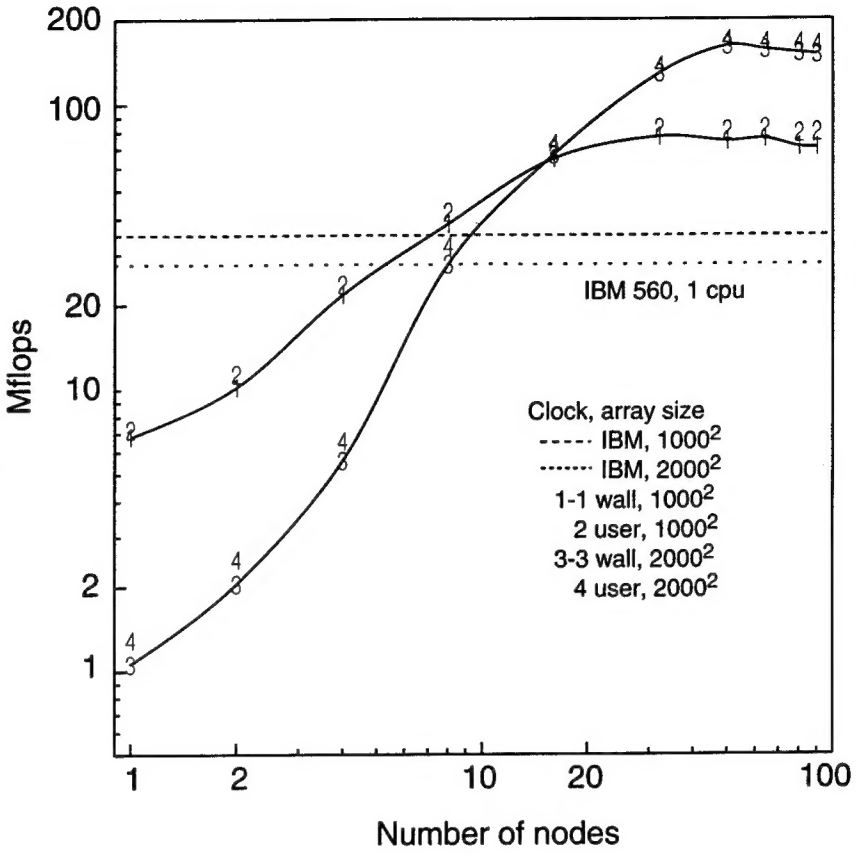

Figure 2. Yee algorithm processor speed. *Wall* and *user* defined in figure 1. For larger array (96 MB), computation is slowed drastically when only a few processors are used. This retardation agrees with case that main array will not fit into local cache of 32 MB of a single node. Also, as loop workload increases, a peak of high processor efficiency shifts toward more nodes.

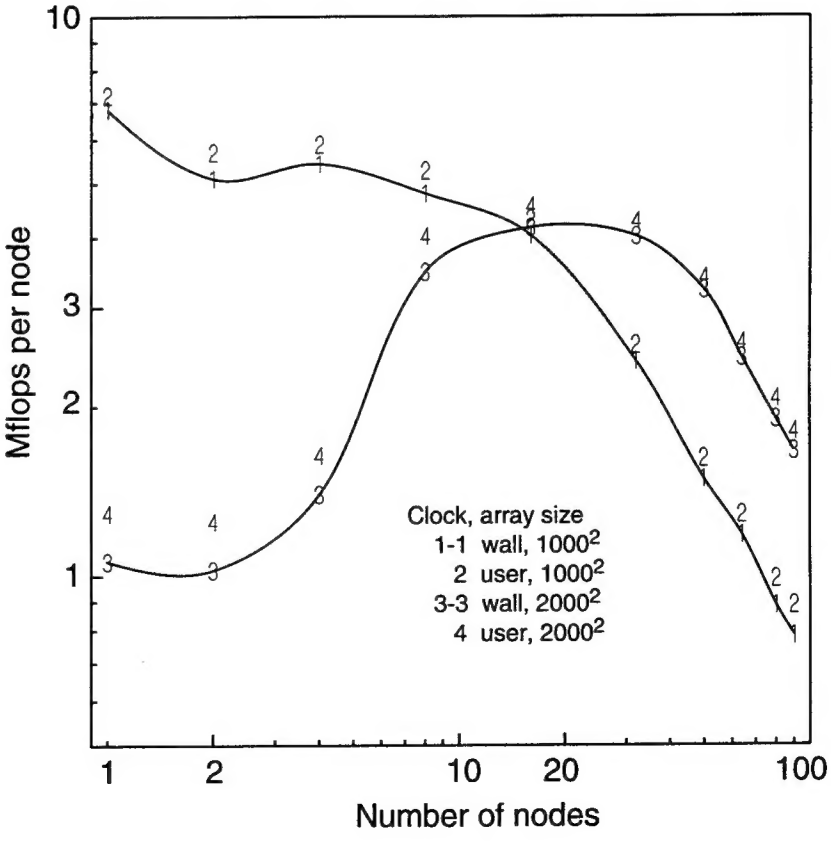

Figure 2 also suggests that if the main loops are given more work, then a higher efficiency will be seen when more nodes are used. In order to exploit this in practice, one would have to take care that the program's memory requirements—that is, largely through its arrays—were not too large for the selected KSR machine.

Scalability from a parallel computing point of view means that the computational throughput (e.g., in Mflops) of an algorithm increases in direct proportion to the number of processors used. The computational speed of the Yee algorithm scales roughly in proportion to the number of nodes used for less than about 20 nodes. With the larger array size, the scalability of the algorithm extends a little further. Thus, the algorithm scales well when it uses no more than about 10 nodes. Above about 10 or 20 nodes, the overhead of the additional nodes, compared with the amount of work they have to do, reduces the returns. Thus, total computational throughput peaks at less than the number of nodes available on the KSR1 machine. For an array of 2000×2000, this peak computing efficiency is reached at about 50 nodes.

An increase in the available work or array size would probably induce a small rise in the peak computing speed. Also, the capacity of the ARL KSR machines for significantly larger problems is probably not there, and would, thus, not allow a substantial further rise in the peak, anyway. This issue is revisited in section 4.2.

4. Comparisons

Several other computing tests helped to further explore the KSR1 capability. One test was a pseudo Yee algorithm and three others were matrix multiply tests. Also, the Yee algorithm and two BLAS matrix multiply tests were run on an IBM RS/6000 model 560.

4.1 Pseudo Yee Algorithm

The pseudo Yee algorithm was identical to the FORTRAN of the Yee algorithm, but without the tiling directives that forced the KSR FORTRAN compiler to separate the *E* and *B* spatial loops. Tiling is a mechanism by which the KSR transforms a single do loop into parallel execution of multiple tiles, or groups of loop execution. When compiled for parallel execution without the tiling loops, the compiler joined the *E* and *B* spatial loops. Of course, this joining gives wrong answers to Maxwell's equations; however, it did enable the KSR to deliver up to about 580 Mflops for a spatial array of 2000×2000. The reason for this boosted speed is probably the reduction in the tile creation overhead, compared with the amount of work a tile set performed. A single tile set could do both *E* and *B*, and it had to be created only once rather than at every time step as the true Yee algorithm had to. This boosted speed does show that the KSR can do some computations substantially faster than the other computations done in this study.

4.2 Matrix Multiplication Speed

Among the standard techniques for solving electromagnetic problems without time-varying parameters are the moment method techniques [6,7]. Solutions to these kind of problems extensively use linear algebra techniques. In an attempt to get a rough estimate of how fast the KSR might solve these kinds of problems, a simple matrix multiply was timed on the KSR1.

The testing program called the subroutine sgemm from the KSR BLAS library to execute the matrix multiply. Since the KSR was using 8-byte reals, sgemm was practically executing in double precision. In one test a matrix of 1000×1000 was multiplied by another matrix of 1000×1000. Matrices of sizes 2000×2000, and 4000×4000, respectively, were multiplied in additional tests. (The KSR UNIX command "unlimit" allowed execution of the largest problems in this study.) Eiffel appeared to have some trouble running the latter program and could not run matrices of size $10{,}000 \times 10{,}000$ or $8{,}000 \times 8{,}000$ in this problem. This problem may also have been a BLAS problem. The apparent limitation in the capacity of Eiffel limited attempts to find scalability beyond 32 or 64 nodes.

Figure 3 shows the computational speed in megaflops as a function of the number of nodes for the matrix multiply operations. Scalability extends up to about 32 nodes, with the larger matrix problems showing scalability over a greater range than for the smaller matrices. Bumps on the curves

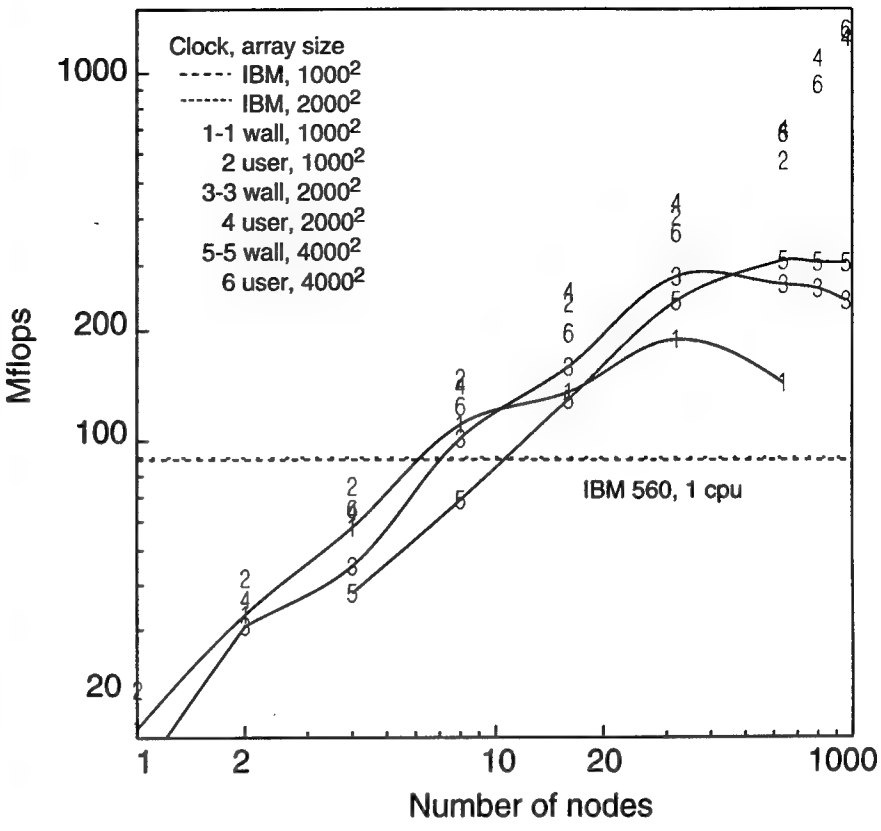

Figure 3. Matrix multiplication speed. IBM, wall, and user defined in figure 1. Square matrices of indicated sizes were multiplied by each other using BLAS subroutines sgemm and dgemm on KSR and IBM machines, respectively. Default single precision for KSR is real*8. Peak matrix multiply speeds of 188.9, 278.8, and 307.1 Mflops occur with 32, 32, and 64 nodes, respectively, based on elapsed wall clock time. Corresponding sizes of multiplied matrices were 1000 × 1000, 2000 × 2000, and 4000 × 4000, respectively.

seem to suggest that the programs optimized performance relatively better at certain numbers of nodes, such as 8 and 32. Except for the 4000 × 4000 matrix, which delivered a peak performance on Eiffel at 64 nodes of 307 Mflops, the other two matrix multiplies peaked at 32 nodes. The data retrieved from the computer runs seem to suggest that matrix multiply performance peaks at about 64 nodes. Whether this is the peak for larger matrix sizes or not may be moot, because of the difficulty in running larger problems.

4.3 Comparison with IBM RS/6000 Performance

IBM RS/6000 workstations have become common high-performance computing platforms in many scientific and engineering facilities throughout the U.S. The above Yee algorithms and the matrix multiplies were run on an IBM RS/6000 model 560 with a 50-MHz clock and 256 Mbytes of RAM for comparison purposes. On the IBM model 560, the 1000 x 1000 and 2000 × 2000 array Yee algorithms ran at 35.0 and 27.7 Mflops in double precision, respectively. Also, on the IBM, the BLAS matrix multiplies using dgemm for the 1000 × 1000 and the 2000 × 2000 matrices ran at 88.7 and 89.8 Mflops, respectively.

5. Conclusions

The Yee algorithm improved by about 2.2 to 5.8 times on the KSR1 compared to our IBM model 560 workstation. The study also suggested that increasing the loop workload through more computations could improve upon this performance. However, port of a full EMP code to the KSR1 would introduce many much slower computations, which the KSR handles poorly. These additional computations would reduce the KSR1 advantage compared to the IBM workstation considerably.

A BLAS matrix multiply improved by only about two or three times on the KSR1. A full moment method code would require a considerable amount of development work for even a modest performance gain. As discussed, optimization of code for the KSR required considerable work for our simple 2-D Yee algorithm.

The KSR1 is significantly less available than our IBM RS/6000 workstations. System crashes and downtime are far more common than on our workstations. Also, on the ARL KSR1 there is much more competition for its large number of processors, which are necessary for high speed.

The disadvantages of the KSR1 are the difficulty in creating high-speed code and its unreliable availability. Its advantage is, at best, only a modest performance gain for our applications in electromagnetism. Rapid improvements in processor performance make or will soon make alternative scalar workstations with equal or better performance available. Because the KSR1 does not provide a net advantage, this study recommends that alternative computers with better performance be used for new EMP codes.

6. References

1. K. S. Yee, "Numerical Solution of Initial Boundary Value Problems Involving Maxwell's Equations in Isotropic Media," *IEEE Transactions on Antennas and Propagation*, **AP-14**, 302–307 (May 1966).

2. R. Holland, "THREDE: A Free-Field EMP Coupling and Scattering Code," *IEEE Trans. Nucl. Sci.*, **NS-24**, No. 6, 2416–2421 (December 1977).

3. W. T. Wyatt, Jr., *Neutron-Induced EMP for a High-Altitude Burst*, Harry Diamond Laboratories, HDL-TR-2205 (February 1992).

4. C. S. Kenyon and W. T. Wyatt, Jr., "Terrain Enhancement of Tactical Source-Region EMP," *Journal of Radiation Effects*, **12**, No. 1 (January 1994).

5. *KSR Fortran Programming Manual*, Kendal Square Research Corporation, 2–5 (July 1993).

6. R. F. Harrington, *Field Computation by Moment Methods*, Macmillan, New York (1968).

7. J. J. H. Wang, *Generalized Moment Methods in Electromagnetics*, Wiley, New York (1991).

Appendix—FORTRAN Source Code

Source code for test of the 2-D Yee algorithm with a 2000 × 2000 size array:

Source Listing for *yee.f.*

```
C 2 Dimensional Yee algorithm for the KSR1
C Compiled under KSR operating system (eiffel KSR OS R1.1.4.1)
C
C Consolidation of field components into a single "X" array was used
C to speed execution on the KSR1:
C X(1,I,J) are the B-phi elements BP(I,J)
C X(2,I,J) are the E-x elements EX(I,J)
C X(3,I,J) are the E-z elements EZ(I,J)
c
 PARAMETER(NK=3,IK=2000,JK=2000,NTMAX=151)
 DIMENSION X(NK,IK,JK),Y(NTMAX,7)
C
C***KSR-SPECIFIC CALLS FOLLOW*****************************
C
 character*14 envvar, envthreads
 parameter (ck=50e-9) !time conversion -> seconds

 include "/usr/include/ksr/pmon.fh"
      integer*8 ibuff(32)
c*ksr* subpage ibuff
 integer istatus

cksr go get the 'PL_NUM_THREADS' environmental variable and convert it
cksr ...to numerical form. (See also KSR Fortran manual chap 13 on getenv)
 envvar = 'PL_NUM_THREADS'
       call getenv(envvar,envthreads)
       decode (14, 104, envthreads,iostat=istat, err=101) nthreads
 104 format (i12)
       print *, 'no error ', 'istat=', istat
 goto 103

 101 write (*,102) envthreads, nthreads,'istat=', istat
 102 format('decode error:envthreads='a14,' nthreads=',i3,' istat=',i2)

 103 print *, 'preyee.f: nthreads = ', nthreads
 call pmon_delta (ibuff,32,istatus)
C
C***END KSR-SPECIFIC CALLS ***************************
c
C INITIALIZATION OF FIELD ARRAYS
 DO 100 J=1,JK
 DO 100 I=2,IK
 DO 100 N=1,NK
 X(N,I,J) = 0
C X(N,1,J) = 0
 100 CONTINUE
C
C Static Boundary Conditions
```

```
C
C Assume the ground is a perfect conductor.
C Zero out horizontal E field (Ex) at ground.
 DO 110 I=1,IK
 X(2,I,1) = 0.0
 110 CONTINUE
C
C Do the vacuum boundary conditions (upper and outer) on the magnetic
c field only.
c First, do the top:
 DO 120 I=1,IK
 X(1,I,JK) = 0.0
 X(1,I,JK-1) = X(1,I,JK)
 120 CONTINUE
C Finally do the outer wall:
 DO 130 J=1,JK
 X(1,IK,J) = 0.0
 130 CONTINUE
C
C Initial Boundary Condition on Ez at x=0
C This effectively sends a step pulse down the mesh
C
 DO 140 J=1,JK
 X(3,1,J) = 10.
 140 CONTINUE
c
 PI = 4*ATAN(1.)
 U0 = PI*4E-7
 C = 2.997925E8
 E0 = 1/(U0*C*C)
 DZ = .25
 DX = .6
C COURANT STABILITY CONDITION:
 DT = .999 * 1/C * 1/SQRT(1/(DX*DX) + 1/(DZ*DZ))
C DT is about 7.69E-10
C
 E21= DT/(DZ*E0)
 E22= E21
 E31= DT/(DX*E0)
 E32= E31
 B1 = DT/(U0*DZ)
 B2 = B1
 B3 = DT/(U0*DX)
 B4 = B3
C
C
C***KSR-SPECIFIC TIMING CALLS FOLLOW*****************************
C
C** WRITE TIMES FOR INITIALIZATION LOOPS
 call pmon_delta (ibuff,ipmsize,istatus)
 write(*,22)
 22 format (13x,'usrclock walclock CEUstallT sp_miss sp_miss_t',
 1 ' dsc_miss')
 write(*,20) 'iniloop ',ibuff(1)*ck,ibuff(2)*ck,ibuff(4)*ck,
```

```
 1 ibuff(10),ibuff( 11)*ck,ibuff(12)
 20 format (a10, 1x, 3f9.3, i11, f9.3, i11)
C
C***END KSR-SPECIFIC TIMING CALLS ****************************
C X(1,I,J) are the B-phi elements BP(I,J)
C X(2,I,J) are the E-x elements EX(I,J)
C X(3,I,J) are the E-z elements EZ(I,J)
C
C
 DO 300 NT=1,NTMAX
C
C
C N TO N+1
C
C*KSR* TILE ( J,I,ORDER=( J,I ) )
 DO 200 J=2,JK-1
 DO 200 I=2,IK-1
       X(2,I,J)= X(2,I,J)-E21*X(1,I,J+1)+E22*X(1,I,J)
       X(3,I,J)= X(3,I,J)+E31*X(1,I+1,J)-E32*X(1,I,J)
 200 CONTINUE
C*KSR* END TILE
C
C N TO N+1
C*KSR* TILE ( J,I,ORDER=( J,I ) )
 DO 250 J=2,JK-1
 DO 250 I=2,IK-1
       X(1,I,J)= X(1,I,J)-B1*X(2,I,J)+B2*X(2,I,J-1)-
 " B3*X(3,I-1,J)+B4*X(3,I,J)
 250 CONTINUE
C*KSR* END TILE
C
C save selected data points
 Y(NT,1)=X(1,5,5)
 Y(NT,2)=X(2,5,5)
 Y(NT,3)=X(3,5,5)
 Y(NT,4)=X(1,5,6)
 Y(NT,5)=X(1,6,5)
 Y(NT,6)=X(2,5,4)
 Y(NT,7)=X(3,4,5)
C
 300 CONTINUE
C
C***KSR-SPECIFIC TIMING CALLS FOLLOW****************************
C
C** WRITE TIMES FOR MAIN LOOPS
 call pmon_delta (ibuff,ipmsize,istatus)
 write(*,20) 'mainloop ',ibuff(1)*ck,ibuff(2)*ck,ibuff(4)*ck,
 1 ibuff(10),ibuff( 11)*ck,ibuff(12)
 call pmon_print (ibuff,32)
C
C***END KSR-SPECIFIC TIMING CALLS ****************************
C
C DO 400 NT = 1,NTMAX
 DO 400 NT = 1,51
```

```
 WRITE (*,310)
 310 FORMAT(' NT, X(1,5,5),X(2,5,5),X(3,5,5),X(1,5,6),X(1,6,5),',
 " 'X(2,5,4),X(3,4,5)')
 WRITE(*,320)NT,(Y(NT,I),I=1,7)
 320 FORMAT(1X,I4,7F9.6,/)
 400 CONTINUE
 STOP
 END
```

Source code listing for BLAS matrix-matrix multiply test with 2000 × 2000 sized matrices:

Source Listing for *matrixm2.f.*

```
 dimension a(2000,2000),b(2000,2000),cout(2000,2000)
 character*1 n,c
C
C***KSR-SPECIFIC CALLS FOLLOW*****************************
C
 character*14 envvar, envthreads
 parameter (ck=50e-9) !time conversion -> seconds

 include "/usr/include/ksr/pmon.fh"
 integer*8 ibuff(32)
c*ksr* subpage ibuff
 integer istatus

cksr go get the 'PL_NUM_THREADS' environmental variable and convert it
cksr ...to numerical form. (See also KSR Fortran manual chap 13 on getenv)
 envvar = 'PL_NUM_THREADS'
 call getenv(envvar,envthreads)
 decode (14, 104, envthreads,iostat=istat, err=101) nthreads
 104 format (i12)
 print *, 'no error ', 'istat=', istat
 goto 103

 101 write (*,102) envthreads, nthreads,'istat=', istat
 102 format('decode error:envthreads='a14,' nthreads=',i3,' istat=',i2)

 print *, '2000x2000 matrix multiply'
 103 print *, 'matrixm2.f: nthreads = ', nthreads
 call pmon_delta (ibuff,32,istatus)
C
C***END KSR-SPECIFIC CALLS ******************************
c
c
c initialialize a and b matrices
 do 10 i = 1,2000
 do 10 j = 1,2000
 a(i,j) = .1 + i + j
 b(i,j) = .1 + 2 * (i+j)
 10 continue
```

```
C
C***KSR-SPECIFIC TIMING CALLS FOLLOW*****************************
C
C** WRITE TIMES FOR INITIALIZATION LOOPS
 call pmon_delta (ibuff,ipmsize,istatus)
 write(*,22)
 22 format (13x,'usrclock walclock CEUstallT sp_miss sp_miss_t',
 1 ' dsc_miss')
 write(*,20) 'iniloop ',ibuff(1)*ck,ibuff(2)*ck,ibuff(4)*ck,
 1 ibuff(10),ibuff( 11)*ck,ibuff(12)
 20 format (a10, 1x, 3f9.3, i11, f9.3, i11)
C
C***END KSR-SPECIFIC TIMING CALLS ******************************

c
c Do Blas matrix - matrix multiply a * b'
 call sgemm ('n','c',2000,2000,2000,1.,a,2000,b,2000,0,cout,2000)
C
C***KSR-SPECIFIC TIMING CALLS FOLLOW*****************************
C
C** WRITE TIMES FOR MAIN LOOPS
 call pmon_delta (ibuff,ipmsize,istatus)
 write(*,20) 'mainloop ',ibuff(1)*ck,ibuff(2)*ck,ibuff(4)*ck,
 1 ibuff(10),ibuff( 11)*ck,ibuff(12)
 call pmon_print (ibuff,32)
C
C***END KSR-SPECIFIC TIMING CALLS ******************************
C
 print *,cout(2,2),cout(46,46)
 stop
 end
```

Distribution

Admnstr
Defns Techl Info Ctr
Attn DTIC-DDA (2 copies)
Cameron Sta Bldg 5
Alexandria VA 22304-6145

Director
Defns Intllgnc Agcy
Attn RTS-2A Techl Lib
Washington DC 20301

Defns Nuc Agcy
Attn RAAE Atmospheric Effects Div
B Prasad
Attn RAEE Elect Effects Div
Attn RAEV Electromagnetic Applctn Div
Attn TISI-Scntfc Info Div
6801 Telegraph Rd
Alexandria VA 22310-3398

Commander
Atmospheric Sci Lab
Attn STEWS-NE J Meason
Attn Techl Lib
White Sands Missile Range NM 88002-5030

Dpty Dir for Rsrch & Engrg (Sci & Techlgy)
Attn COL J Butt
Room 3E118 Pentagon
Washington DC 20301-3080

Ofc of the Assist Scy of the Army for Rsrch
Dev & Acqstn
Attn SARD-TT Dr F Milton
Attn SARD-TT C Nash
Rm 3E479 The Pentagon
Washington DC 20310-0103

Commander
U.S. Army Nuc & Chem Agcy
Attn MONA-NU A Renner
Attn MONA-NU R Pfeffer
7150 Heller Loop Rd Ste 101
Springfield VA 22150-3198

US Army Matl Cmnd
Attn AMCAM-CN
5001 Eisenhower Ave
Alexandria VA 22333-0001

Director
US Army Mis Cmnd (USAMICOM)
Attn AMSMI-RD-CS-R Documents
Redstone Arsenal AL 35898-5400

Director
US Army TMDE Actvty
Attn AMXTM-S-A C Bosco
Redstone Arsenal AL 35898-5400

Nav Rsrch Lab
Attn 4200 Ctr for Sp Sensing
Attn 4600 Condensed Matter & Radiation Sci
Attn 4700 Plasma Physics Div
Attn 5700 Tactical Elect Warfare Div
Attn 6800 Elect Sci & Techlgy Div
Attn 8000 Ctr for Sp Techlgy
Attn Code 4820 Techl Info Div
4555 Overlook Ave SW
Washington DC 20375-5000

Commander
Nav Surfc Weapons Ctr
Attn Code E231 Techl Lib
Dahlgren VA 22448-5020

HQ USAFA/DFSELD
Attn HQUSAFA/DFSELD
2354 Fairchild Dr Ste 3A22
USAF Academy CO 80840-6214

US Air Force Phillips Lab
Attn PL/WSR C Baum
3550 Aberdeen Ave SE
Albuquerque NM 87116-008

Distribution

Lawrence Livermore Natl Lab
Attn L-156 M Bland
Attn L-86 H S Cabayan
PO Box 808
Livermore CA 94550

Sandia Natl Lab
Attn Orgn 3141 Reports Acqstn
PO Box 5800
Albuquerque NM 87185

Natl Inst of Stand & Techlgy
Attn V Ulbrecht Rsrch Info Ctr
Rm E01 Bldg 101
Gaithersburg MD 20899

Univ of Maryland
Attn L Davis
Attn G Sobieski
College Park MD 20742

Univ of Minnesota
Army High Peform Computing Rsrch Ctr
Attn T E Tezduyar
1100 Washington Ave S
Minneapolis MN 55415

Electromagnetic Applctn Inc
Attn R Perala
PO Box 26
Denver CO 80227

Elite Commctn Inc
Attn W T Wyatt
4200 Daniels Ave, Ste 101
Annandale VA 22003

Kendall Square Rsrch Corp
Attn Techl Lib
170 Tracer Lane
Waltham MA 02154-1379

Mssn Rsrch Corp
Attn J Dando
Attn W Stark
4935 N 30th Stret
Colorado Springs CO 80919-3156

Instit for Advncd Techlgy
Attn Techl Lib
4020 2 West Breaker
Austin TX 78759

US Army Rsrch Lab
Attn AMSRL-SL-CS J Beilfuss
Edgewood MD 21010-5423

US Army Rsrch Laboratory
Attn AMSRL-SL-C S Share
Aberdeen MD 21005

US Army Rsrch Lab
Attn AMSRL-CI-CA M Coleman
Attn AMSRL-SL-I W Hughes
Attn AMSRL-CI-A H J Breaux
Attn AMSRL-CI-AC R Sheroke
Attn AMSRL-CI-AD C Ellis
Attn AMSRL-CI-AD C Nietubicz
Attn AMSRL-CI-C W Sturek
Attn AMSRL-CI-CA N Patel
Attn AMSRL-CI-S R Pearson
Attn AMSRL-CI-SA T Kendall
Attn AMSRL-SL-CO A Van de Wal
Attn AMSRL-SL-CM D Farenwald
Aberdeen Proving Ground MD 21005-5068

US Army Rsrch Lab
Attn AMSRL-SL-CM M Mar
Attn AMSRL-WT-NC R Lottero
Aberdeen Proving Ground MD 21005-5425

US Army Rsrch Lab
Attn AMSRL-SL-E Chf
White Sands Missile Range NM 88002-5513

US Army Rsrch Lab
Attn AMSRL-OP-SD-TA Mail & Records Mgmt
Attn AMSRL-OP-SD-TL Tech Library (3 copies)
Attn AMSRL-OP-SD-TP Tech Pub
Attn AMSRL-SS Chf Scientist
Attn AMSRL-SS-S Chf
Attn AMSRL-SS-SI Chf

Distribution

US Army Rsrch Lab (cont'd)
Attn AMSRL-WT-N J Ingram
Attn AMSRL-WT-N J McGarrity
Attn AMSRL-WT-NA R A Kehs
Attn AMSRL-WT-NB J Gwaltney
Attn AMSRL-WT-ND B Luu
Attn AMSRL-WT-ND J R Miletta
Attn AMSRL-WT-ND C S Kenyon
(10 copies)

US Army Rsrch Lab (cont'd)
Attn AMSRL-WT-ND R J Chase
Attn AMSRL-WT-NF L Jasper
Attn AMSRL-WT-NG T Oldham
Attn AMSRL-WT-NH J Corrigan
Attn AMSRL-WT-NH M Bushell